

# Improved Algorithms for Multiple Sink Location Problems in Dynamic Path Networks

Yuuya Higashikawa<sup>1</sup>, Mordecai J. Golin<sup>2</sup>, and Naoki Katoh<sup>1</sup> \*

<sup>1</sup> Department of Architecture and Architectural Engineering, Kyoto University, Japan, {as.higashikawa, naoki}@archi.kyoto-u.ac.jp

<sup>2</sup> Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, golin@cs.ust.hk

**Abstract.** This paper considers the  $k$ -sink location problem in dynamic path networks. In our model, a dynamic path network consists of an undirected path with positive edge lengths, uniform edge capacity, and positive vertex supplies. Here, each vertex supply corresponds to a set of evacuees. Then, the problem requires to find the optimal location of  $k$  sinks in a given path so that each evacuee is sent to one of  $k$  sinks. Let  $\mathbf{x}$  denote a  $k$ -sink location. Under the optimal evacuation for a given  $\mathbf{x}$ , there exists a  $(k - 1)$ -dimensional vector  $\mathbf{d}$ , called  $(k - 1)$ -divider, such that each component represents the boundary dividing all evacuees between adjacent two sinks into two groups, i.e., all supplies in one group evacuate to the left sink and all supplies in the other group evacuate to the right sink. Therefore, the goal is to find  $\mathbf{x}$  and  $\mathbf{d}$  which minimize the maximum cost or the total cost, which are denoted by the minimax problem and the minisum problem, respectively. We study the  $k$ -sink location problem in dynamic path networks with continuous model, and prove that the minimax problem can be solved in  $O(kn)$  time and the minisum problem can be solved in  $O(n^2 \cdot \min\{k, 2^{\sqrt{\log k \log \log n}}\})$  time, where  $n$  is the number of vertices in the given network. Note that these improve the previous results by [6].

**Keywords:** sink location, dynamic network, evacuation planning

## 1 Introduction

The Tohoku-Pacific Ocean Earthquake happened in Japan on March 11, 2011, and many people failed to evacuate and lost their lives due to severe attack by tsunamis. From the viewpoint of disaster prevention from city planning and evacuation planning, it has now become extremely important to establish effective evacuation planning systems against large scale disasters. In particular, arrangements of tsunami evacuation buildings in large Japanese cities near the coast has become an urgent issue. To determine appropriate tsunami evacuation buildings, we need to consider where evacuation buildings are assigned and how

---

\* Supported by JSPS Grant-in-Aid for Scientific Research(A)(25240004)

to partition a large area into small regions so that one evacuation building is designated in each region. This produces several theoretical issues to be considered. Among them, this paper focuses on the location problem of multiple evacuation buildings assuming that we fix the region such that all evacuees in the region are planned to evacuate to one of these buildings. In this paper, we consider the simplest case for which the region consists of a single road.

In order to represent the evacuation, we consider the *dynamic* setting in graph networks, which was first introduced by Ford et al. [3]. In a graph network under the dynamic setting, each vertex is given supply and each edge is given length and capacity which limits the rate of the flow into the edge per unit time. We call such networks under the dynamic setting *dynamic networks*. Dynamic networks can be considered in discrete and continuous models. In discrete model, each input value is given as an integer. Then each supply can be regarded as a set of evacuees, and edge capacity is defined as the maximum number of evacuees who can enter an edge per unit time. On the other hand, in continuous model, each input value is given as a real number. Then each supply can be regarded as fluid, and edge capacity is defined as the maximum amount of supply which can enter an edge per unit time. In either model, we assume that all supply at a vertex is sent to the same sink. The *k-sink location problem in dynamic networks* is defined as the problem which requires to find the optimal location of  $k$  sinks in a given network so that all supply of each vertex is sent to one of  $k$  sinks in the shortest time.

For the 1-sink location problem in dynamic networks, the following two criteria can be naturally considered: *maximum cost criterion* and *total cost criterion* (in static networks, these criteria correspond to the center problem and the median problem in facility location, respectively). If a sink location  $x$  is given in a dynamic network with discrete model, the cost of  $x$  for an evacuee is defined as the minimum time required to send him/her to  $x$  (by taking into account the congestion). Then two criteria are defined as the maximum of cost of  $x$  for all evacuees and the sum of cost of  $x$  for all evacuees, respectively. Now let us turn to continuous model. In continuous model, we define the *unit* as the infinitesimally small portion of supply, then the cost is defined on each unit. If a sink location  $x$  is given in a dynamic network with continuous model, the cost of  $x$  for a unit is defined as the minimum time required to send the unit to  $x$ . Also two criteria are defined as the maximum of cost of  $x$  for all units and the sum of cost of  $x$  for all units, respectively. Definitions for *k-sink location problem in dynamic networks* are given later. Then, the *minimax* (resp. *minisum*) *k-sink location problem in dynamic networks* requires to find a  $k$ -sink location in a given dynamic network which minimizes the maximum (resp. total) cost. Mamada et al. [8] studied the minimax 1-sink location problem in dynamic tree networks with discrete model assuming that the sink must be located at a vertex, and proposed an  $O(n \log^2 n)$  time algorithm. Higashikawa et al. [5] also studied the same problem as [8] assuming that edge capacity is uniform and the sink can be located at any point in the network, and proposed an  $O(n \log n)$  time algorithm. Recently, Higashikawa et al. [6] studied the  $k$ -sink location problems in a dynamic path network with

continuous model assuming that edge capacity is uniform and the sink can be located at any point in the network, and proved that the minimax problem can be solved in  $O(kn \log n)$  time and the minisum problem can be solved in  $O(kn^2)$  time.

In this paper, we study the same problems as [6], and improve the previous time bounds:  $O(kn \log n)$  to  $O(kn)$  for the minimax problem and  $O(kn^2)$  to  $O(n^2 \cdot \min\{k, 2^{\sqrt{\log k \log \log n}}\})$  for the minisum problem.

## 2 Minimax $k$ -sink location problem

### 2.1 Preliminaries

**Model definition:** Let  $P = (V, E)$  be an undirected path where  $V = \{v_1, v_2, \dots, v_n\}$  and  $E = \{e_1, e_2, \dots, e_{n-1}\}$  such that  $v_i$  and  $v_{i+1}$  are endpoints of  $e_i$  for  $1 \leq i \leq n-1$ . Let  $\mathcal{N} = (P, l, w, c, \tau)$  be a dynamic network with the underlying graph being a path  $P$ ,  $l$  is a function that associates each edge  $e_i$  with positive length  $l_i$ ,  $w$  is also a function that associates each vertex  $v_i$  with positive weight  $w_i$  representing the amount of supply at  $v_i$ ,  $c$  is a positive constant representing the amount of supply which can enter an edge per unit time, and  $\tau$  is also a constant representing the time required by flow for traversing the unit distance. We call such networks with path structures *dynamic path networks*. In the following, we use the notation  $P$  to denote the set of all points  $p \in P$ . Also, for a vertex  $v_i \in P$  with  $1 \leq i \leq n$ , we abuse the notation  $v_i$  to denote the distance from  $v_1$  to  $v_i$ , and for a point  $p \in P$ , we abuse the notation  $p$  to denote the distance from  $v_1$  to  $p$ . Then, we can regard  $P$  as embedded on a real line such that  $v_1 = 0$ . For two points  $p, q \in P$  with  $p < q$ , let  $[p, q]$  (resp.  $[p, q)$ ,  $(p, q]$  and  $(p, q)$ ) denote the part of  $P$  which consists of all points  $x \in P$  such that  $p \leq x \leq q$  (resp.  $p \leq x < q$ ,  $p < x \leq q$  and  $p < x < q$ ).

**$k$ -sink location and  $(k-1)$ -divider:** Suppose that  $k$  sinks are located at points  $x_1, x_2, \dots, x_k \in P$  such that  $x_1 \leq x_2 \leq \dots \leq x_k$ , respectively. Note that each sink can be located at any point in  $P$ . In this paper, we assume that if we place a sink at a vertex, all supply of the vertex can finish the evacuation in no time. So, without loss of generality, we assume  $k \leq n$  (otherwise, at least one sink can be located at each vertex). Let  $\mathbf{x} = (x_1, x_2, \dots, x_k)$  which is a  $k$ -dimensional vector, called  *$k$ -sink location*. Let us consider the optimal evacuation for a given  $\mathbf{x}$ . In this paper, we assume that all units of a vertex are sent to the same sink. We call a directed path along which all units of a vertex are sent to a sink *evacuation path*. Then, any two evacuation paths never cross each other in an optimal evacuation (otherwise, we can realize the better or equivalent evacuation by exchanging the two destinations of crossing evacuation paths). Suppose that there exists only one vertex  $v_j$  in  $[x_i, x_{i+1}]$  and all units of the vertex are sent to  $x_i$ , then  $x_{i+1}$  can be moved to  $v_{j+1}$  without increasing the cost of any unit. Therefore, if we optimally locate  $k$  sinks with  $k \geq 2$ , there exist at least two vertices in  $[x_i, x_{i+1}]$  for any  $i$  with  $1 \leq i \leq k-1$ , i.e., there exist two vertices

$v_j$  and  $v_{j+1}$  with  $1 \leq j \leq n-1$  in  $[x_i, x_{i+1}]$  such that all supplies on  $[x_i, v_j]$  are sent to  $x_i$  and all supplies on  $[v_{j+1}, x_{i+1}]$  are sent to  $x_{i+1}$ . We call such a vertex  $v_j$  *dividing vertex*. For an integer  $i$  with  $1 \leq i \leq k-1$  with  $k \geq 2$ , let  $d_i$  be an index of the dividing vertex in  $[x_i, x_{i+1}]$ . By the above discussion,  $d_{i-1} + 1 \leq d_i$  holds for  $1 \leq i \leq k$  where  $d_0 = -1$  and  $d_k = n$ . Let  $\mathbf{d} = (d_1, d_2, \dots, d_{k-1})$  which is a  $(k-1)$ -dimensional vector, called  $(k-1)$ -*divider*. For a given  $\mathbf{d}$ , we need only consider  $\mathbf{x}$  such that  $x_i$  is given on  $[v_{d_{i-1}+1}, v_{d_i}]$  for  $1 \leq i \leq k$ , where  $d_0 = 0$  and  $d_k = n$ .

**Problem definition:** For given  $\mathbf{x}$  and  $\mathbf{d}$ , and also for an integer  $i$  with  $1 \leq i \leq k$ , let  $\Theta_i(\mathbf{x}, \mathbf{d})$  denote the minimum time required to send all supplies on  $[v_{d_{i-1}+1}, v_{d_i}]$  to  $x_i$ , where  $d_0 = 0$  and  $d_k = n$ . Letting  $\Theta(\mathbf{x}, \mathbf{d}) = \max\{\Theta_i(\mathbf{x}, \mathbf{d}) \mid 1 \leq i \leq k\}$ , the minimax  $k$ -sink location problem is defined as follows:

$$Q_{\text{minimax}}(P) : \text{minimize } \{\Theta(\mathbf{x}, \mathbf{d}) \mid \mathbf{x} \in P^k \text{ and } \mathbf{d} \in \{1, 2, \dots, n\}^{k-1}\}. \quad (1)$$

## 2.2 Recursive formulation

We now consider a subproblem of the above mentioned problem: for some integers  $i, j$  and  $p$  with  $1 \leq i \leq j \leq n$  and  $1 \leq p \leq k$ , the  $p$ -sink location problem in  $[v_i, v_j]$ . For  $[v_i, v_j]$ , let  $\mathbf{x}^*(p, i, j)$  denote the optimal  $p$ -sink location and  $\mathbf{d}^*(p, i, j)$  denote the optimal  $(p-1)$ -divider. Note that  $\mathbf{x}^*(p, i, j)$  is a  $p$ -dimensional vector and  $\mathbf{d}^*(p, i, j)$  is also a  $(p-1)$ -dimensional vector, so  $\mathbf{d}^*(p, i, j)$  is not defined for  $p = 1$ . Also, let  $\text{OPT}(p, i, j)$  denote the optimal cost of  $p$ -sink location in  $[v_i, v_j]$ , i.e., the minimum time required to send all supplies on  $[v_i, v_j]$  divided by  $\mathbf{d}^*(p, i, j)$  to  $\mathbf{x}^*(p, i, j)$ . Note that if  $p \geq j - i + 1$  holds, the optimal sink location is trivial, i.e.,  $\text{OPT}(p, i, j) = 0$ .

Next, we show the recursive formula of  $\text{OPT}(p, i, j)$ . For integers  $i, j$  and  $p$  with  $1 \leq i \leq j \leq n$  and  $1 \leq p \leq k-1$ , let us consider the optimal  $(p+1)$ -sink location and  $p$ -divider for  $[v_i, v_j]$ , i.e.,  $\mathbf{x}^*(p+1, i, j)$  and  $\mathbf{d}^*(p+1, i, j)$ . Since any two evacuation paths never cross each other in an optimal evacuation, there exists an integer  $h$  with  $i \leq h \leq j-1$  such that all supplies on  $[v_{h+1}, v_j]$  are sent to the rightmost sink and all supplies on  $[v_i, v_h]$  are sent to the other  $k$  sinks. Thus, we have the following recursion:

$$\text{OPT}(p+1, i, j) = \min_{i \leq h \leq j-1} \max\{\text{OPT}(p, i, h), \text{OPT}(1, h+1, j)\}. \quad (2)$$

Here, let  $d$  be an integer which minimizes the maximum of  $\text{OPT}(p, i, h)$  and  $\text{OPT}(1, h+1, j)$  on  $i \leq h \leq j-1$ :

$$d = \underset{i \leq h \leq j-1}{\text{argmin}} \max\{\text{OPT}(p, i, h), \text{OPT}(1, h+1, j)\}. \quad (3)$$

Then,  $\mathbf{x}^*(p+1, i, j)$  and  $\mathbf{d}^*(p+1, i, j)$  can be represented by using  $d$  as follows:

$$\mathbf{x}^*(p+1, i, j) = (\mathbf{x}^*(p, i, d), \mathbf{x}^*(1, d+1, j)), \quad (4)$$

$$\mathbf{d}^*(p+1, i, j) = (\mathbf{d}^*(p, i, d), d). \quad (5)$$

### 2.3 Known properties of 1-sink location problem

Here, we introduce the properties of 1-sink location problem, which were explicitly shown in [6] (based on [2,4]). For fixed integers  $i$  and  $j$  with  $1 \leq i \leq j \leq n$ , let us consider how to compute the optimal 1-sink location in  $[v_i, v_j]$ . Suppose that a sink is located at a point  $x$  in  $[v_i, v_j]$ . Let  $\Theta_{i,j}(x)$  denote the minimum time required to send all supplies on  $[v_i, v_j]$  to  $x$ . Here, let  $L_i(x)$  (resp.  $R_j(x)$ ) denote the minimum time required to send all supplies on  $[v_i, x]$  (resp.  $[x, v_j]$ ) to  $x$  where  $L_i(v_i) = 0$  and  $R_j(v_j) = 0$ . Then,  $\Theta_{i,j}(x)$  is the maximum of  $L_i(x)$  and  $R_j(x)$ , i.e.,

$$\Theta_{i,j}(x) = \max\{L_i(x), R_j(x)\}. \quad (6)$$

For discrete model, Kamiyama et al. [7] showed that  $L_i(x)$  and  $R_j(x)$  are expressed as follows:

$$L_i(x) = \max_l \left\{ \tau(x - v_l) + \left\lceil \frac{\sum_{i \leq h \leq l} w_h}{c} \right\rceil - 1 \mid v_l \in [v_i, x] \right\},$$

$$R_j(x) = \max_l \left\{ \tau(v_l - x) + \left\lceil \frac{\sum_{l \leq h \leq j} w_h}{c} \right\rceil - 1 \mid v_l \in (x, v_j] \right\}.$$

From these, we can immediately develop the formulae for continuous model as follows:

$$L_i(x) = \max_l \left\{ \tau(x - v_l) + \frac{\sum_{i \leq h \leq l} w_h}{c} \mid v_l \in [v_i, x] \right\}, \quad (7)$$

$$R_j(x) = \max_l \left\{ \tau(v_l - x) + \frac{\sum_{l \leq h \leq j} w_h}{c} \mid v_l \in (x, v_j] \right\}. \quad (8)$$

Note that  $L_i(x)$  (resp.  $R_j(x)$ ) is a piecewise linear strictly increasing (resp. decreasing) function of  $x$ . Therefore, a function  $\Theta_{i,j}(x)$  is unimodal in  $x$ , and there exists the unique point which minimizes  $\Theta_{i,j}(x)$ , they is,  $\mathbf{x}^*(1, i, j)$ . Then, as [2,4,6] showed, we immediately have the following claim.

**Claim 1** For any integers  $i$  and  $j$  with  $1 \leq i \leq j \leq n$  and a point  $x \in [v_i, v_j]$ ,

- (i) if  $L_i(x) \leq R_j(x)$  holds,  $\mathbf{x}^*(1, i, j) \geq x$  holds, and
- (ii) if  $L_i(x) \geq R_j(x)$  holds,  $\mathbf{x}^*(1, i, j) \leq x$  holds.

In the following, when  $x$  is at a vertex  $v_t$  with  $i \leq t \leq j$ , we use the notation  $L(i, t)$  (resp.  $R(t, j)$ ) to denote the value  $L_i(v_t)$  (resp.  $R_j(v_t)$ ). Then, we have the following claim (which was also shown in [2,4,6]).

**Claim 2** For given integers  $i$  and  $j$  with  $1 \leq i \leq j \leq n$ , suppose that for the interval  $[v_l, v_{l+1}]$  with  $i \leq l \leq j-1$ ,  $L(i, l) \leq R(l, j)$  and  $L(i, l+1) \geq R(l+1, j)$  hold, and let  $\alpha^*$  denote the solution to an equation for  $\alpha$ :  $R(l, j) - \alpha\tau(v_{l+1} - v_l) = L(i, l+1) - (1 - \alpha)\tau(v_{l+1} - v_l)$ . Then,

- (i) if  $1 \leq \alpha^* \leq 1$  holds,  $\mathbf{x}^*(1, i, j)$  is a point dividing the interval  $[v_l, v_{l+1}]$  with the ratio of  $\alpha^*$  to  $1 - \alpha^*$  and  $\text{OPT}(1, i, j) = R(l, j) - \alpha^*\tau(v_{l+1} - v_l)$  holds,
- (ii) if  $\alpha^* < 0$  holds,  $\mathbf{x}^*(1, i, j) = v_l$  and  $\text{OPT}(1, i, j) = R(l, j)$  hold, and
- (iii) if  $\alpha^* > 1$  holds,  $\mathbf{x}^*(1, i, j) = v_{l+1}$  and  $\text{OPT}(1, i, j) = L(i, l+1)$  hold.

## 2.4 Key properties of $k$ -sink location problem

In this section, we show several key properties of the  $k$ -sink location problem. Here, for integers  $p$  and  $i$  with  $2 \leq p \leq k$  and  $2 \leq i \leq n$ , let  $f_{p,i}(t)$  denote a function defined on  $\{t \in \mathbb{Z} \mid 1 \leq t \leq i-1\}$ :

$$f_{p,i}(t) = \max\{\text{OPT}(p-1, 1, t), \text{OPT}(1, t+1, i)\}. \quad (9)$$

Note that for fixed  $p$  and  $i$ ,  $\text{OPT}(p-1, 1, t)$  is monotonically increasing in  $t$  and  $\text{OPT}(1, t+1, i)$  is monotonically decreasing in  $t$ . Thus, we have the following claim.

**Claim 3** *For any integers  $p$  and  $i$  with  $2 \leq p \leq k$  and  $2 \leq i \leq n$ , function  $f_{p,i}(t)$  is unimodal in  $t$  on  $1 \leq t \leq i-1$ .*

Let  $d_{p,i}$  be an integer which minimizes  $f_{p,i}(t)$  for  $1 \leq t \leq i-1$ :

$$d_{p,i} = \underset{1 \leq t \leq i-1}{\operatorname{argmin}} f_{p,i}(t). \quad (10)$$

By Claim 3, there uniquely exists  $d_{p,i}$ . By (4) and (5), we have

$$\mathbf{x}^*(p, 1, i) = (\mathbf{x}^*(p-1, 1, d_{p,i}), \mathbf{x}^*(1, d_{p,i}+1, i)), \quad (11)$$

$$\mathbf{d}^*(p, 1, i) = (\mathbf{d}^*(p-1, 1, d_{p,i}), d_{p,i}). \quad (12)$$

Then, we prove the following two lemmas.

**Lemma 1** *For any integers  $p$  and  $i$  with  $2 \leq p \leq k$  and  $2 \leq i \leq n-1$ ,  $d_{p,i} \leq d_{p,i+1}$  holds.*

**Lemma 2** *For any integers  $h, i, j$  and  $l$  with  $1 \leq i \leq j \leq n$ ,  $1 \leq h \leq l \leq n$ ,  $i \leq h$  and  $j \leq l$ ,  $\mathbf{x}^*(1, i, j) \leq \mathbf{x}^*(1, h, l)$  holds.*

**Proof of Lemma 1:** In order to prove Lemma 1, we first confirm a fundamental property.

**Claim 4** *For any integers  $p$  with  $1 \leq p \leq k$ , and  $h, i, j$  and  $l$  with  $1 \leq h \leq i \leq j \leq l \leq n$ ,  $\text{OPT}(p, i, j) \leq \text{OPT}(p, h, l)$  holds.*

We prove Lemma 1 by contradiction: there exist integers  $p$  and  $i$  with  $2 \leq p \leq k$  and  $2 \leq i \leq n-1$  such that  $d_{p,i} > d_{p,i+1}$  holds. For ease of notation in the proof, we use the notations  $A, B, C, D, E$  and  $F$  as follows:

$$\begin{aligned} A &= \text{OPT}(p-1, 1, d_{p,i}), & B &= \text{OPT}(1, d_{p,i}+1, i), \\ C &= \text{OPT}(p-1, 1, d_{p,i+1}), & D &= \text{OPT}(1, d_{p,i+1}+1, i+1), \\ E &= \text{OPT}(1, d_{p,i+1}+1, i), & F &= \text{OPT}(1, d_{p,i}+1, i+1). \end{aligned} \quad (13)$$

From the assumption of  $d_{p,i} > d_{p,i+1}$  and Claim 4, we can derive the following inequalities:

$$C \leq A, \quad (14)$$

$$B \leq E \leq D, \quad (15)$$

$$B \leq F \leq D. \quad (16)$$

Since  $d_{p,i}$  minimizes  $f_{p,i}(t) = \max\{\text{OPT}(p-1, 1, t), \text{OPT}(1, t+1, i)\}$  (refer to (9) and (10)), we have the following inequality:

$$\max\{A, B\} \leq \max\{C, E\}. \quad (17)$$

Also, without loss of generality, we assume that  $d_{p,i+1}$  is maximized unless the cost increases. By this assumption, we have the following inequality:

$$\max\{C, D\} < \max\{A, F\}. \quad (18)$$

Then, we consider three cases: [Case 1]  $A \leq B$ ; [Case 2]  $D \leq C$ ; [Case 3]  $B < A$  and  $C < D$ .

[Case 1]: By (14), (16) and the condition of  $A \leq B$ , we have  $C \leq A \leq F \leq D$ , which contradicts (18).

[Case 2]: By (14), (15) and the condition of  $D \leq C$ , we have  $B \leq E \leq C \leq A$ . By this and (17), we have  $A \leq C$ . Also, by (14), (16) and the condition of  $D \leq C$ , we have  $F \leq D \leq C \leq A$ . By this and (18), we have  $C < A$ , which contradicts  $A \leq C$ .

[Case 3]: By (17) and the condition of  $B < A$ , we have

$$A \leq \max\{C, E\}. \quad (19)$$

Also, by (18) and the condition of  $C < D$ , we have

$$D < \max\{A, F\}. \quad (20)$$

If  $F \leq A$  holds, we have  $D < \max\{C, E\}$  by (19) and (20), which contradicts the condition of  $C < D$  or (15). If  $A < F$  holds, we have  $D < F$  by (20), which contradicts (16).  $\square$

**Proof of Lemma 2:** In order to prove Lemma 2, we first confirm the following claim (refer to the definitions of (7) and (8)).

**Claim 5** (i) For any integers  $i$  and  $j$  with  $1 \leq j \leq i \leq n$  and any points  $x$  and  $y$  with  $v_i \leq x \leq y \leq v_n$ ,  $L_i(x) \leq L_j(x)$  and  $L_i(x) \leq L_i(y)$  hold.  
(ii) For any integers  $i$  and  $j$  with  $1 \leq i \leq j \leq n$  and any points  $x$  and  $y$  with  $v_1 \leq y \leq x \leq v_i$ ,  $R_i(x) \leq R_j(x)$  and  $R_i(x) \leq R_i(y)$  hold.

We prove Lemma 2 by contradiction: there exist integers  $h, i, j$  and  $l$  with  $1 \leq i \leq j \leq n$ ,  $1 \leq h \leq l \leq n$ ,  $i \leq h$  and  $j \leq l$  such that  $\mathbf{x}^*(1, i, j) > \mathbf{x}^*(1, h, l)$  holds. By this assumption, we have the following inequality:

$$i \leq h \leq \mathbf{x}^*(1, h, l) < \mathbf{x}^*(1, i, j) \leq j \leq l. \quad (21)$$

For ease of notation in the proof, we use the notations  $A, B, C, D, E, F, G$  and  $H$  as follows:

$$\begin{aligned} A &= L_i(\mathbf{x}^*(1, i, j)), \quad B = R_j(\mathbf{x}^*(1, i, j)), \\ C &= L_h(\mathbf{x}^*(1, h, l)), \quad D = R_l(\mathbf{x}^*(1, h, l)), \\ E &= L_i(\mathbf{x}^*(1, h, l)), \quad F = R_j(\mathbf{x}^*(1, h, l)), \\ G &= L_h(\mathbf{x}^*(1, i, j)), \quad H = R_l(\mathbf{x}^*(1, i, j)). \end{aligned} \quad (22)$$

From (21) and Claim 5, we can derive the following inequalities:

$$C \leq E \leq A, \quad (23)$$

$$C \leq G \leq A, \quad (24)$$

$$B \leq F \leq D, \quad (25)$$

$$B \leq H \leq D. \quad (26)$$

Since  $\mathbf{x}^*(1, i, j)$  and  $\mathbf{x}^*(1, h, l)$  are the unique points which minimize  $\Theta_{i,j}(x) = \max\{L_i(x), R_j(x)\}$  and  $\Theta_{h,l}(x) = \max\{L_h(x), R_l(x)\}$ , respectively (refer to (6)), we have the following inequalities:

$$\max\{A, B\} < \max\{E, F\}, \quad (27)$$

$$\max\{C, D\} < \max\{G, H\}. \quad (28)$$

Then, we consider three cases: [Case 1]  $A \leq B$ ; [Case 2]  $D \leq C$ ; [Case 3]  $B < A$  and  $C < D$ .

[Case 1]: By (24), (26) and the condition of  $A \leq B$ , we have  $C \leq G \leq H \leq D$ , which contradicts (28).

[Case 2]: By (23), (25) and the condition of  $D \leq C$ , we have  $B \leq F \leq E \leq A$ , which contradicts (27).

[Case 3]: By (27) and the condition of  $B < A$ , we have

$$A < \max\{E, F\}. \quad (29)$$

Also, by (28) and the condition of  $C < D$ , we have

$$D < \max\{G, H\}. \quad (30)$$

If  $F \leq E$  holds, we have  $A < E$  by (29), which contradicts (23). Also, if  $G \leq H$  holds, we have  $D < H$  by (30), which contradicts (26). If  $E < F$  and  $H < G$  hold, we have  $A < F \leq D < G$  by (25), (29) and (30), that is,  $A < G$  holds, which contradicts (24).  $\square$

## 2.5 Algorithm based on dynamic programming

The algorithm basically computes  $\text{OPT}(1, 1, 1), \dots, \text{OPT}(1, 1, n), \text{OPT}(2, 1, 1), \dots, \text{OPT}(2, 1, n), \dots, \text{OPT}(k, 1, 1), \dots, \text{OPT}(k, 1, n)$  in this order. For some integers  $p$  and  $i$  with  $2 \leq p \leq k$  and  $2 \leq i \leq n$ , let us consider how to obtain  $\text{OPT}(p, 1, i)$ . Actually, in order to obtain  $\text{OPT}(p, 1, i)$ , the algorithm needs  $\text{OPT}(p-1, 1, l)$  for  $l = 1, 2, \dots, n$  and  $\text{OPT}(p, 1, i-1)$ , which are supposed to have been obtained. By (2), (9) and (10), we have

$$\text{OPT}(p, 1, i) = f_{p,i}(d_{p,i}) = \max\{\text{OPT}(p-1, 1, d_{p,i}), \text{OPT}(1, d_{p,i}+1, i)\}. \quad (31)$$

Here, we assumed that  $\text{OPT}(p-1, 1, d_{p,i})$  has already been obtained. Thus, in order to obtain  $\text{OPT}(p, 1, i)$ , we only need to compute  $\text{OPT}(1, d_{p,i}+1, i)$ . Recall that  $d_{p,i}$  is the unique point which minimizes function  $f_{p,i}(t)$  (refer to



(9) and (10)). Now, the algorithm knows where  $d_{p,i-1}$  exists, and by Lemma 1,  $d_{p,i-1} \leq d_{p,i}$  holds. So the algorithm starts to compute  $f_{p,i}(t)$  for  $t = d_{p,i-1}$ , and continues to compute in ascending order of  $t$ , as will be shown below. Note that function  $f_{p,i}(t)$  is unimodal in  $t$  by Claim 3, which implies that  $f_{p,i}(t)$  is strictly decreasing until  $t = d_{p,i}$ . Thus, if the algorithm reaches the first integer  $t^* \geq d_{p,i-1}$  such that  $f_{p,i}(t^*) \leq f_{p,i}(t^* + 1)$ , it outputs  $t^*$  as  $d_{p,i}$ . Then, the algorithm also outputs  $f_{p,i}(t^*)$  as  $\text{OPT}(p, 1, i)$ .

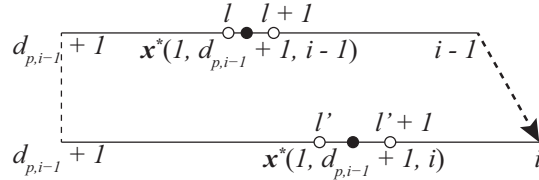
**Computation of  $f_{p,i}(t)$  for  $t = d_{p,i-1}$ :** As above mentioned, the algorithm first computes  $f_{p,i}(t)$  with  $t = d_{p,i-1}$  which is defined as follows:

$$f_{p,i}(d_{p,i-1}) = \max\{\text{OPT}(p-1, 1, d_{p,i-1}), \text{OPT}(1, d_{p,i-1} + 1, i)\}. \quad (32)$$

Since the algorithm has already obtained  $\text{OPT}(p-1, 1, d_{p,i-1})$ , we only need to compute  $\text{OPT}(1, d_{p,i-1} + 1, i)$ . To do this, we actually need to find  $\mathbf{x}^*(1, d_{p,i-1} + 1, i)$ . On the other hand, the algorithm has already obtained  $\text{OPT}(p, 1, i-1)$  as follows:

$$\text{OPT}(p, 1, i-1) = \max\{\text{OPT}(p-1, 1, d_{p,i-1}), \text{OPT}(1, d_{p,i-1} + 1, i-1)\}, \quad (33)$$

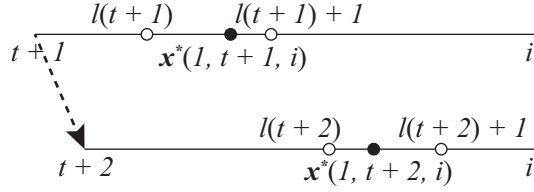
which implies that  $\mathbf{x}^*(1, d_{p,i-1} + 1, i-1)$  has been obtained. By Lemma 2,  $\mathbf{x}^*(1, d_{p,i-1} + 1, i-1) \leq \mathbf{x}^*(1, d_{p,i-1} + 1, i)$  holds. Let  $l$  and  $l'$  be the indices of vertices such that  $\mathbf{x}^*(1, d_{p,i-1} + 1, i-1) \in [v_l, v_{l+1}]$  with  $d_{p,i-1} + 1 \leq l \leq i-2$  and  $\mathbf{x}^*(1, d_{p,i-1} + 1, i) \in [v_{l'}, v_{l'+1}]$  with  $d_{p,i-1} + 1 \leq l' \leq i-1$ , respectively (see Figure 1). By Claim 1, for any interval  $[v_h, v_{h+1}]$  with  $d_{p,i-1} + 1 \leq h \leq i-1$ ,



**Fig. 1.** Illustrations of  $\mathbf{x}^*(1, d_{p,i-1} + 1, i-1)$  and  $\mathbf{x}^*(1, d_{p,i-1} + 1, i)$

there exists  $\mathbf{x}^*(1, d_{p,i-1} + 1, i)$  in  $[v_h, v_{h+1}]$  if  $L(d_{p,i-1} + 1, h) \geq R(h, i)$  and  $L(d_{p,i-1} + 1, h+1) \leq R(h+1, i)$  hold. Therefore, if we maintain the data structure so that we can compute these values, the algorithm can test if there exists  $\mathbf{x}^*(1, d_{p,i-1} + 1, i) \in [v_h, v_{h+1}]$  or not (what the data structure is or how we can maintain and use it will be explained in the next subsection). Then, the algorithm starts to test for  $h = l$ , and continues to test in ascending order of  $h$ . If an interval where  $\mathbf{x}^*(1, d_{p,i-1} + 1, i)$  exists, that is,  $[v_{l'}, v_{l'+1}]$  is found, then  $\mathbf{x}^*(1, d_{p,i-1} + 1, i)$  and  $\text{OPT}(1, d_{p,i-1} + 1, i)$  can be computed in  $O(1)$  time by Claim 2.

**Computation of  $f_{p,i}(t)$  for  $t \geq d_{p,i-1} + 1$ :** Now, suppose that for an integer  $t$  with  $t \geq d_{p,i-1}$ , the algorithm has already obtained  $f_{p,i}(t)$ , that is,  $\mathbf{x}^*(1, t+1, i)$  and  $\text{OPT}(1, t+1, i)$ . For an integer  $t$  with  $t \geq d_{p,i-1}$ , let  $l(t+1)$  be the index of a vertex with  $t+1 \leq l(t+1) \leq i-1$  such that  $\mathbf{x}^*(1, t+1, i) \in [v_{l(t+1)}, v_{l(t+1)+1}]$ . Note that  $l(t+1)$  has also been obtained (see Figure 2). Then, the computation of  $f_{p,i}(t+1)$  comes down to finding  $l(t+2)$  which is greater than or equal to  $l(t+1)$ , and so, it can be treated in the similar manner as the computation of  $f_{p,i}(d_{p,i-1})$ .



**Fig. 2.** Illustrations of  $\mathbf{x}^*(1, t+1, i)$  and  $\mathbf{x}^*(1, t+2, i)$

## 2.6 How to compute $L(\alpha, \beta)$ and $R(\beta, \gamma)$

As mentioned in Section 2.5, in order to obtain  $\text{OPT}(p, 1, i)$  for fixed  $p$  and all  $i = p+1, p+2, \dots, n$  (note that  $\text{OPT}(p, 1, i) = 0$  for  $i = 1, 2, \dots, p$ ), the algorithm computes  $f_{p,p+1}(d_{p,p}), \dots, f_{p,p+1}(d_{p,p+1}), f_{p,p+2}(d_{p,p+1}), \dots, f_{p,p+2}(d_{p,p+2}), \dots, f_{p,n}(d_{p,n-1}), \dots, f_{p,n}(d_{p,n})$ . In this computation, the algorithm actually computes  $L(p, p), L(p, p+1), \dots, L(p, l(p)), L(p+1, l(p)), L(p+1, l(p)+1), \dots, L(p+1, l(p+1)), \dots$  where  $l(i)$  is the index of vertex with  $p \leq l(i) \leq l(i+1) \leq n$  for any  $i \geq p$ , and also,  $R(p, p), R(p, p+1), \dots, R(p, r(p)), R(p+1, r(p)), R(p+1, r(p)+1), \dots, R(p+1, r(p+1)), \dots$  where  $r(i)$  is the index of vertex with  $p \leq r(i) \leq r(i+1) \leq n$  for any  $i \geq p$ . In order to compute  $L(\alpha, \beta)$  and  $R(\beta, \gamma)$  for any integers  $\alpha, \beta$  and  $\gamma$  with  $1 \leq \alpha \leq \beta \leq \gamma \leq n$ , the algorithm maintains the specific data structures  $D_L(\alpha, \beta)$  and  $D_R(\beta, \gamma)$ , respectively. Depending on the situation, the algorithm updates  $D_L(\alpha, \beta)$  to  $D_L(\alpha+1, \beta)$  or  $D_L(\alpha, \beta+1)$  and  $D_R(\beta, \gamma)$  to  $D_R(\beta+1, \gamma)$  or  $D_R(\beta, \gamma+1)$ . We below show definitions of the two data structures and how to maintain these.

**Definition of  $D_L(\alpha, \beta)$  and how to maintain  $D_L(\alpha, \beta)$ :** In this discussion, we assume that  $\alpha < \beta$  holds ( $D_L(\alpha, \beta) = \emptyset$  when  $\alpha = \beta$ ). Let us consider the evacuation of all supplies on  $[v_\alpha, v_\beta]$  to  $v_\beta$ . We define the vertex indices  $\rho_1, \dots, \rho_e$  as

$$\begin{aligned} \rho_1 &= \operatorname{argmax} \left\{ \tau(v_\beta - v_j) + \frac{\sum_{l=\alpha}^j w_l}{c} \mid \alpha \leq j < \beta \right\} \quad \text{and} \\ \rho_i &= \operatorname{argmax} \left\{ \tau(v_\beta - v_j) + \frac{\sum_{l=\rho_{i-1}+1}^j w_l}{c} \mid \rho_{i-1} < j < \beta \right\} \quad \text{for } 2 \leq i \leq e. \end{aligned} \quad (34)$$

Note that  $\rho_e = \beta - 1$  holds. For every integer  $i$  with  $1 \leq i \leq e$ , we also define the value of  $\rho_i$  as  $\sigma_i = \sum \{w_h \mid \rho_{i-1} + 1 \leq h \leq \rho_i\}$  where  $\rho_0 + 1 = \alpha$ . Here, we notice that for every integer  $i$  with  $2 \leq i \leq e$ , the first unit of  $v_{\rho_{i-1}}$  never be induced to stop at any vertex  $v_j$  with  $\rho_{i-1} < j < \beta$ . The data structure  $D_L(\alpha, \beta)$  consists of the two sequences  $(\rho_1, \dots, \rho_e)$  and  $(\sigma_1, \dots, \sigma_e)$ . Note that we define the size of  $D_L(\alpha, \beta)$  as  $|D_L(\alpha, \beta)| = e$ . Recall that in continuous model, the cost is defined on each infinitesimal unit of supply, i.e., the cost of  $x$  for a unit is defined as the minimum time required to send the unit to  $x$ . Here, we notice that for any integer  $i$  with  $2 \leq i \leq e$ , the first unit of  $v_{\rho_{i-1}}$  never be induced to stop at  $v_{\rho_i}$ . Then,  $L(\alpha, \beta)$  can be computed as

$$L(\alpha, \beta) = \tau(v_\beta - v_{\rho_1}) + \frac{\sigma_1}{c}. \quad (35)$$

In order to update  $D_L(\alpha, \beta)$  to  $D_L(\alpha + 1, \beta)$ , the algorithm tests if  $\rho_1 = \alpha$  holds or not. If it holds, the algorithm sets  $D_L(\alpha + 1, \beta)$  so that

$$\rho_i \leftarrow \rho_{i+1} \text{ and } \sigma_i \leftarrow \sigma_{i+1} \text{ for } 1 \leq i \leq e - 1. \quad (36)$$

Otherwise, the algorithm sets  $D_L(\alpha + 1, \beta)$  so that

$$\begin{aligned} \rho_1 &\leftarrow \rho_1 \text{ and } \sigma_1 \leftarrow \sigma_1 - w_\alpha, \\ \rho_i &\leftarrow \rho_i \text{ and } \sigma_i \leftarrow \sigma_i \quad \text{for } 2 \leq i \leq e. \end{aligned} \quad (37)$$

On the other hand, in order to update  $D_L(\alpha, \beta)$  to  $D_L(\alpha, \beta + 1)$ , the algorithm first sets  $\rho_{e+1} = \beta$  and  $\sigma_{e+1} = w_\beta$ . Then, the algorithm repeatedly tests if  $\tau(v_{\rho_{j+1}} - v_{\rho_j}) \leq \sigma_{j+1}/c$  holds or not in descending order of  $j$  from  $j = e$ . If it holds, the algorithm sets  $D_L(\alpha, \beta + 1)$  so that

$$\begin{aligned} \rho_i &\leftarrow \rho_i \text{ and } \sigma_i \leftarrow \sigma_i & \text{for } 1 \leq i \leq j - 1, \\ \rho_j &\leftarrow \rho_{j+1} \text{ and } \sigma_j \leftarrow \sigma_j + \sigma_{j+1}, \end{aligned} \quad (38)$$

until  $\tau(v_{\rho_{e'+1}} - v_{\rho_{e'}}) > \sigma_{e'+1}/c$  holds for  $j = e'$  with some integer  $e' \leq e$ . Let  $t(\alpha, \beta)$  denote the number of such tests required to update  $D_L(\alpha, \beta)$  to  $D_L(\alpha, \beta + 1)$ , which can be represent as

$$t(\alpha, \beta) = e - e' + 1 = |D_L(\alpha, \beta)| - |D_L(\alpha, \beta + 1)| + 2. \quad (39)$$

Recall that in the computation to obtain  $\text{OPT}(p, 1, i)$  for fixed  $p$  and all  $i = p + 1, p + 2, \dots, n$  for a given integer  $\alpha$  with  $p \leq \alpha \leq n - 1$ , the algorithm updates  $D_L(\alpha, l(\alpha - 1))$  to  $D_L(\alpha, l(\alpha))$  where  $l(p - 1) = p$  and  $l(n) = n$ . Let  $T(\alpha)$  denote the total number of such tests required to update  $D_L(\alpha, l(\alpha - 1))$  to  $D_L(\alpha, l(\alpha))$ , and  $T$  denote the sum of  $T(\alpha)$  for  $p \leq \alpha \leq n - 1$ . By (36) and (37), we have  $|D_L(\alpha, l(\alpha))| \geq |D_L(\alpha + 1, l(\alpha))|$ , so the upper bound of  $T$  can be

obtained as

$$\begin{aligned}
T &= \sum_{\alpha=p}^{n-1} T(\alpha) = \sum_{\alpha=p}^{n-1} \sum_{\beta=l(\alpha-1)}^{l(\alpha)} t(\alpha, \beta) \\
&= \sum_{\alpha=p}^{n-1} \{|D_L(\alpha, l(\alpha-1))| - |D_L(\alpha, l(\alpha))| + 2l(\alpha) - 2l(\alpha-1)\} \\
&\leq \sum_{\alpha=p}^{n-1} \{|D_L(\alpha, l(\alpha-1))| - |D_L(\alpha+1, l(\alpha))| + 2l(\alpha) - 2l(\alpha-1)\} \\
&= |D_L(p, l(p-1))| + 2l(n-1) - 2l(p-1) \in O(n-p), \tag{40}
\end{aligned}$$

which implies that  $t(\alpha, \beta)$  is amortized  $O(1)$ .

**Definition of  $D_R(\beta, \gamma)$  and how to maintain  $D_R(\beta, \gamma)$ :** In this discussion, we assume that  $\beta < \gamma$  holds ( $D_R(\beta, \gamma) = \emptyset$  when  $\beta = \gamma$ ). Let us consider the evacuation of all supplies on  $[v_\beta, v_\gamma]$  to  $v_\beta$ . We define the vertex indices  $\mu_1, \dots, \mu_f$  as

$$\begin{aligned}
\mu_1 &= \operatorname{argmax} \left\{ \tau(v_j - v_\beta) + \frac{\sum_{l=j}^{\gamma} w_l}{c} \mid \beta < j \leq \gamma \right\} \quad \text{and} \\
\mu_i &= \operatorname{argmax} \left\{ \tau(v_j - \mu_{i-1}) + \frac{\sum_{l=j}^{\gamma} w_l}{c} \mid \mu_{i-1} < j \leq \gamma \right\} \text{ for } 2 \leq i \leq f. \tag{41}
\end{aligned}$$

Note that  $\mu_f = \gamma$  holds. For every integer  $i$  with  $1 \leq i \leq f$ , we also define the value of  $\mu_i$  as  $W_i = \sum \{w_h \mid \mu_i \leq h \leq n\}$ . Here, we notice that  $v_{\mu_i}$  is the rightmost vertex of which the first unit never be induced to stop at any vertex  $v_j$  with  $\mu_{i-1} < j < \mu_i$  where  $\mu_0 = \beta$ . In addition, let  $os(\gamma) = \sum \{w_h \mid \gamma+1 \leq h \leq n\}$ . The data structure  $D_R(\beta, \gamma)$  consists of the offset value  $os(\gamma)$  and the two sequences  $(\mu_1, \dots, \mu_f)$  and  $(W_1, \dots, W_f)$ . Note that we define the size of  $D_R(\beta, \gamma)$  as  $|D_R(\beta, \gamma)| = f$ . Then,  $R(\beta, \gamma)$  can be computed as

$$R(\beta, \gamma) = \tau(v_{\mu_1} - v_\beta) + \frac{W_1 - os(\gamma)}{c}. \tag{42}$$

In order to update  $D_R(\beta, \gamma)$  to  $D_R(\beta+1, \gamma)$ , the algorithm tests if  $\mu_1 = \beta+1$  holds or not. If it holds, the algorithm sets  $D_R(\beta+1, \gamma)$  so that

$$\mu_i \leftarrow \mu_{i+1} \quad \text{and} \quad W_i \leftarrow W_{i+1} \quad \text{for } 1 \leq i \leq f-1. \tag{43}$$

Otherwise, nothing changes, that is, the algorithm sets  $D_R(\beta+1, \gamma) = D_R(\beta, \gamma)$ .

On the other hand, in order to update  $D_R(\beta, \gamma)$  to  $D_R(\beta, \gamma+1)$ , the algorithm first sets  $\mu_{f+1} = \gamma+1$  and compute  $W_{f+1} = W_f - w_\gamma$  and  $os(\gamma+1) = os(\gamma) - w_{\gamma+1}$ . Then, the algorithm repeatedly tests if  $\tau v_{\gamma+1} + w_{\gamma+1}/c \geq \tau v_{\mu_j} + (W_j - os(\gamma+1))/c$  holds or not in descending order of  $j$  from  $j = f$ . If it holds, the algorithm sets  $D_R(\beta, \gamma+1)$  so that

$$\begin{aligned}
\mu_i &\leftarrow \mu_i \quad \text{and} \quad W_i \leftarrow W_i \quad \text{for } 1 \leq i \leq j-1, \\
\mu_j &\leftarrow \mu_{j+1} \quad \text{and} \quad W_j \leftarrow W_{j+1}, \tag{44}
\end{aligned}$$

until  $\tau v_{\gamma+1} + w_{\gamma+1}/c < \tau v_{\mu_{f'}} + (W_{f'} - os(\gamma + 1))/c$  holds for  $j = f'$  with some integer  $f' \leq f$ . Let  $t'(\beta, \gamma)$  denote the number of such tests required to update  $D_R(\beta, \gamma)$  to  $D_R(\beta, \gamma + 1)$ , which can be represent as

$$t'(\beta, \gamma) = f - f' + 1 = |D_R(\beta, \gamma)| - |D_R(\beta, \gamma + 1)| + 2, \quad (45)$$

which is amortized  $O(1)$  by the same discussion as that for  $t(\alpha, \beta)$  defined at (39).

**Claim 6** *For any integers  $\alpha, \beta$  and  $\gamma$  with  $1 \leq \alpha \leq \beta \leq \gamma \leq n$ ,  $L(\alpha, \beta)$  and  $R(\beta, \gamma)$  can be computed in  $O(1)$  time once  $D_L(\alpha, \beta)$  and  $D_R(\beta, \gamma)$  have been obtained.*

**Claim 7** (i) *For any integers  $\alpha, \beta$  and  $\gamma$  with  $1 \leq \alpha < \beta < \gamma \leq n$ ,  $L(\alpha, \beta)$  and  $R(\beta, \gamma)$  can be updated to  $L(\alpha + 1, \beta)$  and  $R(\beta + 1, \gamma)$  in amortized  $O(1)$  time, respectively.*

(ii) *For any integers  $\alpha, \beta$  and  $\gamma$  with  $1 \leq \alpha \leq \beta \leq \gamma \leq n - 1$ ,  $L(\alpha, \beta)$  and  $R(\beta, \gamma)$  can be updated to  $L(\alpha, \beta + 1)$  and  $R(\beta, \gamma + 1)$  in amortized  $O(1)$  time, respectively.*

## 2.7 Time complexity

As mentioned in Section 2.5 and at the beginning of Section 2.6, in order to obtain  $\text{OPT}(p, 1, i)$  for fixed  $p$  and all  $i = p + 1, p + 2, \dots, n$ ,  $O(n)$  intervals are tested in total as follows: in order to test if there exists  $\mathbf{x}^*(1, i, j)$  in an interval  $[v_h, v_{h+1}]$  or not, the algorithm needs to confirm that  $L(i, h) \geq R(h, j)$  and  $L(i, h + 1) \leq R(h + 1, j)$  hold by Claim 1, which takes  $O(1)$  time once  $D_L(i, h)$ ,  $D_L(i, h + 1)$ ,  $D_R(h, j)$  and  $D_R(h + 1, j)$  have been obtained by Claim 6. Thus, such computations take  $O(n)$  time in total.

On the other hand, let us consider the total time required to update the data structures. For fixed  $p$  and  $i$ , when  $\text{OPT}(p, 1, i - 1)$  is obtained, the algorithm maintains  $D_L(d_{p, i-1} + 1, l)$ ,  $D_L(d_{p, i-1} + 1, l + 1)$ ,  $D_R(l, i - 1)$  and  $D_R(l + 1, i - 1)$ , where  $\mathbf{x}^*(1, d_{p, i-1} + 1, i - 1)$  exists in  $[v_l, v_{l+1}]$ . When  $\text{OPT}(p, 1, i)$  is obtained after repeatedly updating these four vertex sets, the algorithm maintains  $D_L(d_{p, i} + 1, l')$ ,  $D_L(d_{p, i} + 1, l' + 1)$ ,  $D_R(l', i)$  and  $D_R(l' + 1, i)$ , where  $\mathbf{x}^*(1, d_{p, i} + 1, i)$  exists in  $[v_{l'}, v_{l'+1}]$ . Recall that  $d_{p, i-1} \leq d_{p, i}$  and  $l \leq l'$  hold by Lemmas 1 and 2. Thus, in order to obtain  $\text{OPT}(p, 1, i)$ , the algorithm updates the four vertex sets  $2(d_{p, i} - d_{p, i-1}) + 4(l' - l) + 2$  times, and so, for fixed  $p$  and all  $i = 1, 2, \dots, n$ , the algorithm updates these sets  $O(n)$  times in total, which takes  $O(n)$  time by Claim 7.

Therefore,  $\text{OPT}(p, 1, i)$  for all  $i = 1, 2, \dots, n$  and  $p = 1, 2, \dots, k$  can be obtained in  $O(kn)$  time.

**Theorem 1** *The minimax  $k$ -sink location problem in a dynamic path network with uniform capacity can be solved in  $O(kn)$  time.*

### 3 Minisum $k$ -sink location problem

In this section, an input graph of this problem is a dynamic path network defined in Section 2. As a preliminary step, let us consider the minisum 1-sink location problem.

#### 3.1 Properties of the minisum 1-sink location problem

Suppose that a sink is located at a point  $x \in P$  where  $P$  is the input path with  $n + 1$  vertices. In continuous model, the cost is defined on each infinitesimal unit of supply, i.e., the cost of  $x$  for a unit is defined as the minimum time required to send the unit to  $x$ . Let  $sum(x)$  denote the total cost of  $x$ , i.e., the sum of cost of  $x$  for all units on  $P$ . Here, let  $sum_L(x)$  (resp.  $sum_R(x)$ ) denote the sum of cost of  $x$  for all units on  $[v_1, x]$  (resp.  $(x, v_n]$ ). Then,  $sum(x)$  is the maximum of  $sum_L(x)$  and  $sum_R(x)$ , i.e.,

$$sum(x) = sum_L(x) + sum_R(x). \quad (46)$$

Without loss of generality, we assume  $sum_L(v_1) = 0$  and  $sum_R(v_n) = 0$ . Now, suppose that  $x$  is located in an open interval  $(v_h, v_{h+1})$  with  $1 \leq h \leq n - 1$ , then let us explain how function  $sum_L(x)$  is determined.

**Case 1:** For every integer  $i$  with  $1 \leq i \leq h$ ,  $\tau(v_i - v_{i-1}) > w_i/c$  holds. In this case, the first unit of each vertex on  $[v_1, v_h]$  can reach  $x$  after leaving the original vertex without being blocked due to the existence of other units at an intermediate vertex. For an integer  $i$  with  $1 \leq i \leq h$ , let  $sum^i(x)$  denote the sum of cost of  $x$  for all units of  $v_i$ . Here, suppose that there are  $\alpha$  units at  $v_i$  with sufficiently large  $\alpha$ , i.e., the size of each unit is equal to  $w_i/\alpha$ , and these units continuously reach  $x$ . Then by (7), the  $l$ -th unit finishes reaching  $x$  at time  $\tau(x - v_i) + l \cdot (w_i/\alpha)/c$ . Therefore, by taking  $\alpha$  to the infinity,  $sum^i(x)$  can be represented as follows:

$$\begin{aligned} sum^i(x) &= \lim_{\alpha \rightarrow \infty} \sum_{l=1}^{\alpha} \frac{w_i}{\alpha} \left( \tau(x - v_i) + l \cdot \frac{w_i}{\alpha} \cdot \frac{1}{c} \right) \\ &= \int_0^1 \left( w_i \tau(x - v_i) + \frac{w_i^2}{c} \cdot r \right) dr = w_i \tau(x - v_i) + \frac{w_i^2}{2c}, \end{aligned} \quad (47)$$

and also  $sum_L(x)$  is represented as follows:

$$sum_L(x) = \sum_{1 \leq i \leq h} sum^i(x) = \sum_{1 \leq i \leq h} \left( w_i \tau(x - v_i) + \frac{w_i^2}{2c} \right). \quad (48)$$

**Case 2:** We define the vertex indices  $\rho_1, \dots, \rho_e$  as

$$\begin{aligned} \rho_1 &= \operatorname{argmax} \left\{ \tau(v_h - v_j) + \frac{\sum_{l=1}^j w_l}{c} \mid 1 \leq j \leq h \right\} \quad \text{and} \\ \rho_i &= \operatorname{argmax} \left\{ \tau(v_h - v_j) + \frac{\sum_{l=\rho_{i-1}+1}^j w_l}{c} \mid \rho_{i-1} < j \leq h \right\} \text{ for } 2 \leq i \leq e. \end{aligned} \quad (49)$$

Note that  $\rho_e = h$  holds. For every integer  $i$  with  $1 \leq i \leq e$ , we also define the value of  $\rho_i$  as  $\sigma_i = \sum \{w_h \mid \rho_{i-1} + 1 \leq h \leq \rho_i\}$  where  $\rho_0 = 0$ . Here, we notice that for every integer  $i$  with  $2 \leq i \leq e$ , the first unit of  $v_{\rho_{i-1}}$  never be induced to stop at any vertex  $v_j$  with  $\rho_{i-1} < j < \beta$ . Then, as with (48),  $\operatorname{sum}_L(x)$  is represented as follows:

$$\operatorname{sum}_L(x) = \sum_{1 \leq i \leq e} \left( \sigma_i \tau(x - \rho_i) + \frac{\sigma_i^2}{2c} \right). \quad (50)$$

Note that  $\sum_{1 \leq i \leq h^*} \sigma_i = \sum_{1 \leq i \leq h} w_i$  holds.

We can compute  $\operatorname{sum}_R(x)$  in the similar manner as  $\operatorname{sum}_L(x)$ . Thus, for an open interval  $(v_j, v_{j+1})$  with  $1 \leq j \leq n-1$ , function  $\operatorname{sum}(x)$  is linear in  $x$  with slope  $\tau(\sum_{1 \leq i \leq j} w_i - \sum_{j+1 \leq i \leq n} w_i)$ . Now let us consider an open interval  $(v_j, v_{j+1})$  with  $1 \leq j \leq n-1$  such that  $\sum_{1 \leq i \leq j} w_i - \sum_{j+1 \leq i \leq n} w_i \geq 0$  holds. Then, we can see that for any two points  $p, q \in (v_j, v_{j+1})$  with  $p < q$ ,  $\operatorname{sum}(p) \leq \operatorname{sum}(q)$  holds. We will show that for sufficiently small  $\epsilon > 0$ ,  $\operatorname{sum}(v_j) \leq \operatorname{sum}(v_j + \epsilon)$  holds. We confirm

$$\operatorname{sum}_R(v_j) = \operatorname{sum}_R(v_j + \epsilon) + \left( \sum_{j+1 \leq i \leq n} w_i \right) \cdot \tau \epsilon, \quad \text{and} \quad (51)$$

$$\operatorname{sum}_L(v_j + \epsilon) \geq \operatorname{sum}_L(v_j) + \left( \sum_{1 \leq i \leq j} w_i \right) \cdot \tau \epsilon. \quad (52)$$

From (51), (52) and the assumption of  $\sum_{1 \leq i \leq j} w_i - \sum_{j+1 \leq i \leq n} w_i \geq 0$ , we can derive  $\operatorname{sum}(v_j) \leq \operatorname{sum}(v_j + \epsilon)$ . In general, we have the following claim.

**Claim 8** (i) For an open interval  $(v_j, v_{j+1})$  with  $1 \leq j \leq n-1$  such that  $\sum_{1 \leq i \leq j} w_i - \sum_{j+1 \leq i \leq n} w_i \geq 0$ ,  $\operatorname{sum}(v_j) \leq \operatorname{sum}(p)$  holds where  $p \in (v_j, v_{j+1})$ .  
(ii) For an open interval  $(v_j, v_{j+1})$  with  $1 \leq j \leq n-1$  such that  $\sum_{1 \leq i \leq j} w_i - \sum_{j+1 \leq i \leq n} w_i < 0$ ,  $\operatorname{sum}(v_{j+1}) < \operatorname{sum}(p)$  holds where  $p \in (v_j, v_{j+1})$ .

Let  $x^*$  denote the optimal sink location which minimizes  $\operatorname{sum}(x)$ . Then, Claim 8 implies that  $x^*$  is located at some vertex.

**Claim 9** There exists  $x^*$  at a vertex.

### 3.2 Algorithm and time complexity for the minisum 1-sink location problem

We propose the algorithm which can solve the minisum 1-sink location problem in a dynamic path network. Basically, the algorithm first computes  $sum_L(v_i)$  for  $2 \leq i \leq n$  in ascending order of  $i$ , and next  $sum_R(v_i)$  for  $1 \leq i \leq n-1$  in descending order of  $i$ . After computing all these values,  $sum(v_i)$  can be computed and evaluated for  $1 \leq i \leq n$  in  $O(n)$  time. Then, by Claim 9, the optimal sink location  $x^*$  is at a vertex which minimizes  $sum(v_i)$  for  $1 \leq i \leq n$ . Below, we show how to compute  $sum_L(v_i)$  (computation of  $sum_R(v_i)$  can be treated in the similar manner).

First, the algorithm sets  $\rho_1 = 1$ ,  $\sigma_1 = w_1$ . By (48),  $sum_L(v_2)$  is computed in  $O(1)$  time as follows:

$$sum_L(v_2) = \sigma_1 \tau(v_2 - v_{\rho_1}) + \frac{\sigma_1^2}{2c}. \quad (53)$$

Now, suppose that for some integer  $j$  with  $1 \leq j \leq n-1$ ,  $h(j)$  has been set as a non-negative integer,  $\rho_i$  and  $\sigma_i$  have been obtained for all  $i$  with  $1 \leq i \leq h(j)$  in the same manner as mentioned in Case 2, Section 3.1, and  $sum_L(v_j)$  has been already computed as follows:

$$sum_L(v_j) = \sum_{1 \leq i \leq h(j)} \left( \sigma_i \tau(v_j - v_{\rho_i}) + \frac{\sigma_i^2}{2c} \right). \quad (54)$$

Let  $W_{j-1} = \sum_{1 \leq i \leq j-1} w_i = \sum_{1 \leq i \leq h(j)} \sigma_i$  and suppose that  $W_{j-1}$  has also been computed. We then show how to compute  $sum_L(v_{j+1})$ . The algorithm newly sets

$$sum' = sum_L(v_j), \quad \text{and} \quad W' = W_{j-1}. \quad (55)$$

Next, the algorithm tests if  $\tau(v_j - v_{\rho_i}) \leq w_j/c$  for  $1 \leq i \leq h(j)$  in descending order. If so, it updates  $sum'$  and  $W'$  as follows:

$$sum' \leftarrow sum' - \left( \sigma_i \tau(v_j - v_{\rho_i}) + \frac{\sigma_i^2}{2c} \right), \quad \text{and} \quad W' \leftarrow W' - \sigma_i, \quad (56)$$

and deletes  $\rho_i$ . If the maximum integer  $m$  such that  $\tau(v_j - v_{\rho_m}) > w_j/c$  is found or  $\tau(v_j - v_{\rho_1}) \leq w_j/c$  is obtained, the algorithm stops testing. In the former case, after the algorithm tests  $h(j) - m + 1$  times,  $\rho_1, \dots, \rho_m$  remain. Then, after computing  $W_j$  as  $W_j = W_{j-1} + w_j$ , by (50),  $sum_L(v_{j+1})$  can be computed as

$$sum_L(v_{j+1}) = sum' + W' \tau(v_{j+1} - v_j) + \left( (W_j - W') \tau(v_{j+1} - v_j) + \frac{(W_j - W')^2}{2c} \right). \quad (57)$$

Also, for the next recursive step, the algorithm eventually sets

$$h(j+1) = m+1, \quad \rho_{m+1} = j, \quad \text{and} \quad \sigma_{m+1} = W_j - W'. \quad (58)$$



Since the algorithm tests  $h(j) - m + 1 = h(j) - h(j+1) + 2$  times to compute  $sum_L(v_{j+1})$ , it needs to test  $\sum_{1 \leq i \leq n-1} (h(i) - h(i+1) + 2)$  times to compute  $sum_L(v_i)$  for  $2 \leq i \leq n$ . Here, by  $h(1) = 0$ , we have

$$\sum_{1 \leq i \leq n-1} (h(i) - h(i+1) + 2) = -h(n) + 2(n-1) = O(n). \quad (59)$$

**Lemma 3** *The minisum 1-sink location problem in a dynamic path network with uniform capacity can be solved in  $O(n)$  time.*

### 3.3 Extension to the minisum $k$ -sink location problem

Let  $\mathbf{x} = (x_1, x_2, \dots, x_k)$  representing a  $k$ -sink location given on  $P$  and  $\mathbf{d} = (d_1, d_2, \dots, d_{k-1})$  representing a  $(k-1)$ -divider given on  $P$  (which are defined in the same manner as mentioned in Section 2.1). For a given  $\mathbf{d}$ , we need only consider  $\mathbf{x}$  such that  $x_i$  is given on  $[v_{d_{i-1}+1}, v_{d_i}]$  for  $1 \leq i \leq k$ , where  $d_0 = 0$  and  $d_k = n$ . For given  $\mathbf{x}$  and  $\mathbf{d}$ , and for an integer  $i$  with  $1 \leq i \leq k$ , let  $sum_i(\mathbf{x}, \mathbf{d})$  denote the sum of cost of  $x_i$  for all supplies on  $[v_{d_{i-1}+1}, v_{d_i}]$ . Letting  $sum(\mathbf{x}, \mathbf{d}) = \sum \{sum_i(\mathbf{x}, \mathbf{d}) \mid 1 \leq i \leq k\}$ , the minisum  $k$ -sink location problem is defined as follows:

$$Q_{\text{minisum}}(P) : \text{minimize } \left\{ sum(\mathbf{x}, \mathbf{d}) \mid \mathbf{x} \in P^k \text{ and } \mathbf{d} \in \{1, 2, \dots, n\}^{k-1} \right\} \quad (60)$$

We below show that this problem can be transformed to an equivalent problem, which requires to find the minimum  $k$ -link path in a weighted, complete, directed acyclic graph (DAG) [9]. First, for integers  $i$  and  $j$  with  $1 \leq i < j \leq n+1$ , let  $\text{OPT}(i, j)$  denote the optimal cost for the minisum 1-sink location problem in  $[v_i, v_{j-1}]$ . Let us consider a DAG  $G = (N, A)$  such that  $N = \{u_1, u_2, \dots, u_n, u_{n+1}\}$  and for every vertex pair  $(u_i, u_j)$  with  $1 \leq i < j \leq n+1$ , there exists an edge which is directed from  $u_i$  to  $u_j$  and associated with the weight of  $\text{OPT}(i, j)$ . Then,  $Q_{\text{minisum}}(P)$  is equivalent to a problem requiring to find a path in  $G$  from  $u_1$  to  $u_{n+1}$  which contains exactly  $k$  edges such that the sum of weights is minimized. Schieber [9] showed that this problem can be solved by querying edge weights  $O(n \cdot \min\{k, 2^{\sqrt{\log k \log \log n}}\})$  times if the input DAG satisfies the *concave Monge property*, that is,  $\text{OPT}(i, j) + \text{OPT}(i+1, j+1) \leq \text{OPT}(i+1, j) + \text{OPT}(i, j+1)$  holds for any integers  $i$  and  $j$  with  $1 \leq i+1 < j \leq n$ . Since each weight query takes  $O(n)$  time by Lemma 3, if the concave Monge property is proved,  $Q_{\text{minisum}}(P)$  can be solved in  $O(n^2 \cdot \min\{k, 2^{\sqrt{\log k \log \log n}}\})$  time. Therefore, we prove the following lemma.

**Lemma 4** *For any integers  $i$  and  $j$  with  $1 \leq i+1 < j \leq n$ ,  $\text{OPT}(i, j) + \text{OPT}(i+1, j+1) \leq \text{OPT}(i+1, j) + \text{OPT}(i, j+1)$  holds.*

*Proof.* For integers  $i$  and  $j$  with  $1 \leq i < j \leq n+1$ , and a 1-sink location  $x \in [v_i, v_{j-1}]$ , let  $sum_{i,j}(x)$  denote the sum of cost of  $x$  for all supplies on  $[v_i, v_{j-1}]$ , and let  $sum_L^i(x)$  (resp.  $sum_R^j(x)$ ) denote the sum of cost of  $x$  for all supplies

on  $[v_i, x)$  (resp.  $(x, v_{j-1}]$ ). Also, let  $x^*(i, j) = \operatorname{argmin}\{sum_{i,j}(x) \mid x \in [v_i, v_{j-1}]\}$ . By the definitions, we have

$$sum_{i,j}(x) = sum_L^i(x) + sum_R^j(x), \text{ and} \quad (61)$$

$$\operatorname{OPT}(i, j) = sum_{i,j}(x^*(i, j)). \quad (62)$$

Then, we consider two cases: [Case 1]  $x^*(i+1, j) \leq x^*(i, j+1)$  and [Case 2]  $x^*(i+1, j) > x^*(i, j+1)$ . Here, let us prove only Case 1 (Case 2 can be symmetrically proved). We first show that

$$\begin{aligned} sum_{i,j}(x^*(i+1, j)) - sum_{i+1,j}(x^*(i+1, j)) \\ \leq sum_{i,j+1}(x^*(i, j+1)) - sum_{i+1,j+1}(x^*(i, j+1)). \end{aligned} \quad (63)$$

By (61), the left side of (63) is equal to  $sum_L^i(x^*(i+1, j)) - sum_L^{i+1}(x^*(i+1, j))$  and the right side of (63) is equal to  $sum_L^i(x^*(i, j+1)) - sum_L^{i+1}(x^*(i, j+1))$ . Let  $D = sum_L^{i+1}(x^*(i, j+1)) - sum_L^{i+1}(x^*(i+1, j))$  (clearly  $D > 0$ ), that is,

$$sum_L^{i+1}(x^*(i, j+1)) = sum_L^{i+1}(x^*(i+1, j)) + D. \quad (64)$$

Then, we have

$$sum_L^i(x^*(i, j+1)) \geq sum_L^i(x^*(i+1, j)) + D. \quad (65)$$

By (64) and (65), we obtain

$$\begin{aligned} sum_L^i(x^*(i+1, j)) - sum_L^{i+1}(x^*(i+1, j)) \\ \leq sum_L^i(x^*(i, j+1)) - sum_L^{i+1}(x^*(i, j+1)), \end{aligned} \quad (66)$$

which is equivalent to (63) as mentioned above. On the other hand, by the optimality of  $\operatorname{OPT}(i, j)$  and  $\operatorname{OPT}(i+1, j+1)$ , we have

$$sum_{i,j}(x^*(i+1, j)) \geq \operatorname{OPT}(i, j), \text{ and} \quad (67)$$

$$sum_{i+1,j+1}(x^*(i, j+1)) \geq \operatorname{OPT}(i+1, j+1). \quad (68)$$

Then, by (63), (67), (68) and the definitions of  $sum_{i+1,j}(x^*(i+1, j)) = \operatorname{OPT}(i+1, j)$  and  $sum_{i,j+1}(x^*(i, j+1)) = \operatorname{OPT}(i, j+1)$ , we obtain

$$\operatorname{OPT}(i, j) - \operatorname{OPT}(i+1, j) \leq \operatorname{OPT}(i, j+1) - \operatorname{OPT}(i+1, j+1), \quad (69)$$

which implies that the lemma holds in Case 1.  $\square$

**Theorem 2** *The minisum  $k$ -sink location problem in a dynamic path network with uniform capacity can be solved in  $O(n^2 \cdot \min\{k, 2^{\sqrt{\log k \log \log n}}\})$  time.*

## 4 Conclusion

In this paper, we study the  $k$ -sink location problem in dynamic path networks with continuous model assuming that edge capacity is uniform and sinks can

be located at any point in the network, and prove that the minimax problem can be solved in  $O(kn)$  time and the minisum problem can be solved in  $O(n^2 \cdot \min\{k, 2^{\sqrt{\log k \log \log n}}\})$  time.

On the other hand, we leave as an open problem to reduce the time bound to  $O(kn)$  for the minisum problem, and extend the solvable networks into dynamic path networks with general capacities or more general networks (e.g., trees).

## References

1. D. Chen and R. Chen, “A Relaxation-Based Algorithm for Solving the Conditional  $p$ -Center Problem”, *Operations Research Letters*, 38(3), pp. 215-217, 2010.
2. S.W. Cheng, Y. Higashikawa, N. Katoh, G. Ni, B. Su and Y. Xu, “Minimax Regret 1-Sink Location Problems in Dynamic Path Networks”, *Proc. The 10th Annual Conference on Theory and Applications of Models of Computation (TAMC 2013)*, LNCS 7876, pp. 121-132, 2013.
3. L. R. Ford Jr. and D. R. Fulkerson, “Constructing Maximal Dynamic Flows from Static Flows”, *Operations Research*, 6, pp. 419-433, 1958.
4. Y. Higashikawa, J. Augustine, S.W. Cheng, M.J. Golin, N. Katoh, G. Ni, B. Su and Y. Xu, “Minimax Regret 1-Sink Location Problem in Dynamic Path Networks”, *Theoretical Computer Science*, DOI: 10.1016/j.tcs.2014.02.010, 2014.
5. Y. Higashikawa, M. J. Golin, N. Katoh, “Minimax Regret Sink Location Problem in Dynamic Tree Networks with Uniform Capacity”, *Proc. The 8th International Workshop on Algorithms and Computation (WALCOM 2014)*, LNCS 8344, pp. 125-137, 2014.
6. Y. Higashikawa, M. J. Golin, N. Katoh, “Multiple Sink Location Problems in Dynamic Path Networks”, *Proc. The 10th International Conference on Algorithmic Aspects of Information and Management (AAIM 2014)*, LNCS 8546 (to appear).
7. N. Kamiyama, N. Katoh and A. Takizawa, “An Efficient Algorithm for Evacuation Problem in Dynamic Network Flows with Uniform Arc Capacity”, *IEICE Transactions*, 89-D(8), pp. 2372-2379, 2006.
8. S. Mamada, T. Uno, K. Makino and S. Fujishige, “An  $O(n \log^2 n)$  Algorithm for the Optimal Sink Location Problem in Dynamic Tree Networks”, *Discrete Applied Mathematics*, 154(16), pp. 2387-2401, 2006.
9. B. Schieber, “Computing a Minimum Weight  $k$ -Link Path in Graphs with the Concave Monge Property”, *Journal of Algorithms*, 29(2), pp. 204-222, 1998.